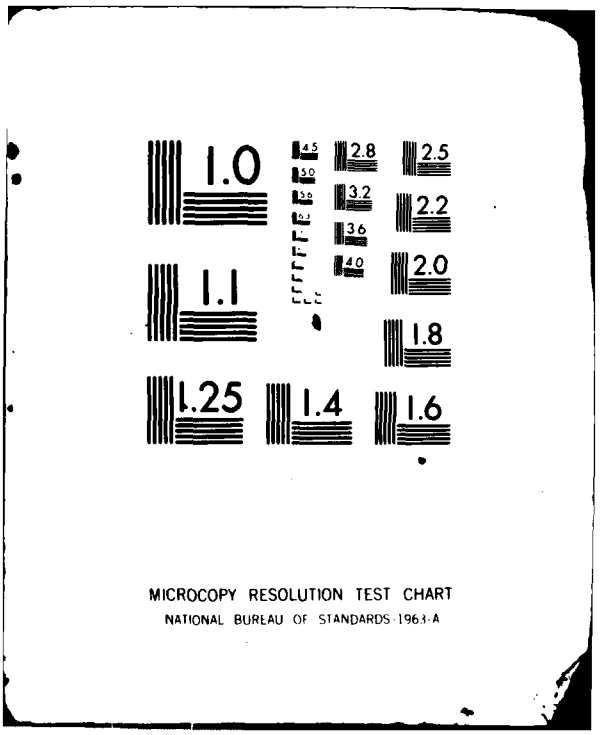
AD-A113 024 STANFORD UNIV CA DEPT OF COMPUTER SCIENCE F/G 9/2
MAXIMAL OBJECTS AND THE SEMANTICS OF UNIVERSAL RELATION DATABAS--ETC(U)
OCT 81 D MAIER, J D ULLMAN AFOSR-80-0212
UNCLASSIFIED STAN-CS-81-878 AFOSR-TR-82-0272 NL

END
DATE FILMED
4-82
DTIC


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6
6.3
7.1
8.0
9.0
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

October 1981

Report. No. STAN-CS-81-878

Maximal Objects and the Semantics of Universal Relation Databases

by

David Maier

Jeffrey D. Ullman

Department of Computer Science

Stanford University
Stanford, CA 94305

Unc l

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER AFOSR-TR- 82-0272	2. GOVT ACCESSION NO. AD-A113 024	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) MAXIMAL OBJECTS AND THE SEMANTICS OF UNIVERSAL RELATION DATABASES		5. TYPE OF REPORT & PERIOD COVERED Interim
		6. PERFORMING ORG. REPORT NUMBER STAN-CS-81-878
7. AUTHOR(s) David Maier, Jeffrey D. Ullman		8. CONTRACT OR GRANT NUMBER(s) AFOSR-80-0212
9. PERFORMING ORGANIZATION NAME AND ADDRESS Stanford University, Dept. of Computer Science, Stanford, CA 94305		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 61102F, 2304/A2
11. CONTROLLING OFFICE NAME AND ADDRESS AFOSR/NM, Bolling AFB, Bldg. #410, Wash. DC 20332		12. REPORT DATE Oct. 1981
		13. NUMBER OF PAGES 10
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

The universal relation concept is intended to provide the database user with a simplified model in which he can compose queries without regard to the underlying structure of the relations in the database. Frequently, the lossless join criterion provides the query interpreter with the clue needed to interpret the query as the user intended. However, some examples exist where interpretation by the lossless join rule runs contrary to our intutition. To handle some of these cases, we propose a concept called maximal objects, which modifies the universal relation concept in exactly those situations where it appears to go

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

Unc l

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

awry-when the underlying relational structure has "cycles". We offer examples of how the maximal object concept provides intuitively correct interpretations. We also consider how one might construct maximal objects mechanically from purely syntactic structural information-the relation schemes and functional dependencies-about the database.

Maximal Objects and the Semantics of Universal Relation Databases

David Maier, SUNY at Stony Brook†
Jeffrey D. Ullman, Stanford University‡

Abstract

The universal relation concept is intended to provide the database user with a simplified model in which he can compose queries without regard to the underlying structure of the relations in the database. Frequently, the lossless join criterion provides the query interpreter with the clue needed to interpret the query as the user intended. However, some examples exist where interpretation by the lossless-join rule runs contrary to our intuition. To handle some of these cases, we propose a concept called *maximal objects*, which modifies the universal relation concept in exactly those situations where it appears to go awry—when the underlying relational structure has "cycles." We offer examples of how the maximal object concept provides intuitively correct interpretations. We also consider how one might construct maximal objects mechanically from purely syntactic structural information—the relation schemes and functional dependencies—about the database.

I. Introduction

We assume the reader is familiar with basic concepts in relational database theory, such as functional and multivalued dependencies, and the operators of relational algebra, particularly projection and (natural) join. [Ul] or [Ma1] contains the needed background.

We also expect the reader is familiar with the notion of a join dependency (JD), which is expressed $\bowtie(R_1, R_2, \ldots, R_n)$ where each $R_i, 1 \leq i \leq n$, is a set of attributes, and is satisfied by a relation r over $R = U_{i=1}^{n} R_i$ if and only if the join of the projections of r onto the R_i's is r itself. Formally:

$$r = \bowtie_{i=1}^{n} \pi_{R_i}(r)$$

A useful notation for JD's was introduced in [FMU]. For $\bowtie(R_1, R_2, \ldots, R_n)$ we construct a hypergraph (graph in which "edges" are arbitrary sets of nodes rather than doubletons only) as follows. For each attribute appearing in one or more of the R_i's the hypergraph has a node. For each R_i, the hypergraph has an edge consisting of all the members of R_i.

Also studied in [FMU] and [BFMY] was a subclass of the JD's—those that have "acyclic" hypoergraphs. The term "acyclic" was given many equivalent definitions in these two papers; here we shall introduce only one. We *Graham-reduce* a hypergraph by applying the following rules in any order (the process is Church-Rosser, so order doesn't really matter).

i) Eliminate a node that appears in only one edge.
ii) Eliminate an edge that is a subset of another edge.

Then a hypergraph (and its JD) is *acyclic* if and only when it reduces to nothing by Graham-reduction.

It is the goal of this paper to contribute to the utility of the "universal relation" view of data. Interestingly, a number of papers have recently been written to argue that the universal relation view is unsupportable for one or anther reason [Kl, BG, AP]. It is not our intent to argue the details of the matters. We shall advance only one argument in its favor: it works; it may not be perfect for everything, but it does certain things well enough to be valued by its users.

In particular, a universal relation system, called System/Q has been operating successfully at Bell Laboratories for some time [A]. It has enabled a number of nontechnical people to use relational database systems with little effort, while just as we would expect, the "experts" must spend considerable effort preparing the system to work on each database. The System/Q approach to query interpretation is to provide a "rel file," which is a list of the sets of relations to join in response to a query, in order of preference. that is, given a query that mentions a set of attributes X, the system goes down the rel file until it finds a set of relation schemes whose union includes X, and it then takes the join of these relations and answers the query as if it were about this join. Further, [S] has recently developed a universal relation system

† Supported in part by NSF grant IST-79-18264.
‡ Supported in part by Air Force grant AFOSR 80-0212 in accordance with NSF agreement IST-80-21358.

that constructs joins in response to queries automatically, based on the theory of functional dependencies and lossless joins. In fact, this strategy of query interpretation bears considerable similarity to the method we propose, but it does not make use of multivalued or join dependencies, or "declared maximal objects," ingredients we consider essential to exploit the power of the universal relation concept. Interestingly, people at Bell have contemplated doing automatic generation of rel files by a method related to that of Sagiv [S].

We begin with the hypothesis of Fagin, Mendelzon and Ullman [FMU], that "real world" universal relations can be described by one join dependency and a collection of functional dependencies. They argue that if the universal relation over a set of attributes $A_1, A_2, \ldots, A_n$ has meaning at all, then we can define it by

$$u = \{ < a_1 a_2 \ldots a_n > | P_1 \wedge P_2 \wedge \ldots \wedge P_k \},$$

where each P_i is a predicate taking some set of the a_j's as arguments (some of these a_j's may be null).

If P_i involves $a_{j_1}, a_{j_2}, \ldots, a_{j_i}$, then the set of attributes $R_i = \{ A_{j_1}, A_{j_2}, \ldots, A_{j_i} \}$ is said to be an *object*; the term "object" corresponds closely to the same term of Sciore [Sc] and is borrowed from there. In essence, objects are sets of attributes among which there is a significant connection. It is proven in [FMU] that u can be constructed in this way if and only if u satisfies the JD $\bowtie(R_1, R_2, \ldots, R_k)$.

Example 1: Suppose our universal relation scheme consists of the attributes

BNK	(bank)
ACC	(account)
L	(loan)
C	(customer)
AMT	(loan amount)
BAL	(account balance)
ADR	(customer address)

We assume the functional dependencies

ACC	→	BNK BAL
L	→	BNK AMT
C	→	ADR

We also assume that the universal relation is defined, in terms of the current "real world facts" as

$$\{ <\text{bnk, acc, l, c, amt, bal, adr}> \mid \text{ACCAT(bnk, acc)} \wedge \text{LAT(bnk, l)} \wedge \text{OWN(acc, c)} \wedge \text{HOLD(l, c)} \wedge \text{HAS(acc, bal)} \wedge \text{FOR(l, amt)} \wedge \text{LIVES(c, adr)} \}$$

where the predicates are defined as

ACCAT(x, y)	=	account y is at bank x
LAT(x, y)	=	loan y is at bank x
OWN(x, y)	=	customer y owns account x
HOLD(x, y)	=	customer y holds loan x
HAS(x, y)	=	account x has balance y
FOR(x, y)	=	loan x is for amount y
LIVES(x, y)	=	customer x lives at address y

Each of these predicates uses knowledge about the present state of the real world to constrain the set of tuples currently appearing in the universal relation. The functional dependencies are facts that we assume are reflected in the predicates. For example, since $C \rightarrow$ ADR holds, we do not expect LIVES(x, y_1) and LIVES(x, y_2) to be true simultaneously if $y_1 \neq y_2$. However, it is possible that the attributes of an FD are contained in no object, in which case the FD is still "true" but its effect is not so easily visible. As should be obvious, the implementation we have in mind for a universal relation

$$u = \{ < a_1 a_2 \ldots a_n > | P_1 \wedge P_2 \wedge \cdots \wedge P_k \}$$

is a database consisting of relations $r_1, r_2, \ldots, r_k$, where r_i is a relation on scheme R_i, and R_i is the object for P_i. (Perhaps some of the relations $r_1, r_2, \ldots, r_k$ are not explicitly stored, but they all can be derived by projection from stored relations.) We have a tuple t_i in r_i exactly when $P_i(t_i)$ is true, so the interpretation

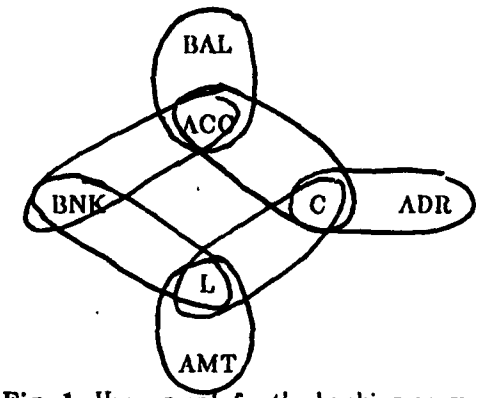


Fig. 1. Hypergraph for the banking example

of each relation should be clear. In the future, we shall identify the relations with the predicates, and talk about relation ACCAT(ACC, BNK), and so on. []

Definition: If $X \subseteq A_1A_2\cdots A_n$, the *connection* among the attributes in X, denoted $[X]$, is defined by

$$[X] = \pi_X(u)$$

That is, $[X]$ is the projection of the universal relation onto the attributes in X.[]

Figure 1 shows the hypergraph representation of the seven objects of our example. In this case, each hyperedge consists of two attributes. This hypergraph has a cycle, which implies that at least one of the attributes is semantically overloaded; it stands for two different things. While we cannot say for sure which attribute is overloaded, since the database is most likely designed from the bank's point of view, we shall consider C overloaded, representing customers in their roles as depositors and borrowers. We shall see in the next section how this overloading causes queries to give an intuitively wrong answer.

II. Queries on the Universal Relation

We shall consider the common form of query on the universal relation that can be expressed by relational operators select, project, and (natural) join. These will be expressed in a QUEL-like notation [SWKH, U1], but without range-statements, since all tuple variables must range over the universal relation. The format of queries we use is

retrieve <attribute list>
where <condition>

The <attribute list> has the form $(t_1.A_1, t_2.A_2, \ldots, t_m.A_m)$, where the t_i's are (not necessarily distinct) tuple variables and the A_i's are (not necessarily distinct) attributes. The <condition> is built from operands that are constants or atoms of the form $t.A$, for tuple variable t and attribute A, using arithmetic comparison $(=, >, \geq, \ldots)$ and Boolean connectives.

The meaning of the query is defined by the following steps:

Algorithm 1:

1. Take the cross product of the universal relation with itself p times, if there are p distinct tuple variables. That is,

   $$E_1 = u \times u \times \cdots \times u \ (p \text{ times})$$

Each copy of u is said to correspond to one particular tuple variable; the correspondence is arbitrary.

2. Replace each occurrence of u by the join $r_1 \bowtie r_2 \bowtie \cdots \bowtie r_k$ of the relations for all the predicates $P_1, P_2, \ldots, P_k$. The result of the substitution is expression E_2. The justification for this substitution is that given a universal relation u exists, and assuming as in [FMU] that the JD $\bowtie(R_1, \ldots, R_k)$ holds, then the result of the join is u. In practice, the r_i's may not be projections of u exactly; there may be "dangling tuples" that do not contribute to the join $r_1 \bowtie \cdots \bowtie r_k$. We shall discuss the significance of this discrepancy shortly.

3. Apply the selection operator for the <condition> in the where-clause to E_2. Every atom $t.A$ of the <condition> refers to a unique component of E_2, the component for attribute A in the copy of the universal relation corresponding to t. Let the result be E_3.
4. Apply the projection operator for the list of components mentioned in the <attribute list> to E_3. The result is an algebraic expression E_4.
5. Optimize the expression under "weak equivalence," that is, find a minimal expression E_5 that is equivalent to E_4 under the assumption that the r_i's are the projections of one universal relation. For expressions of the type we have constructed, assuming reasonable selection conditions, such a minimum always exists ([ASU], [KI]) and can be found efficiently.

Intuitively, the last step throws away terms from the join if they are not necessary to connect one or more attributes in the query. In fact, when (and only when) the hypergraph of the JD defining the universal relation's structure is acyclic, the expression E_5 really does invariably find the minimal lossless join connecting the attributes of the query [MU]. The fact that the expression E_5 involves as few joins as possible has the desirable effect, among others, of ensuring that dangling tuples can contribute to the answer as long as they join successfully with tuples in those of the r_i's that are actually involved in the join.

Example 2: Consider the query on the universal relation of Fig. 1:

retrieve(t.C) (Q_1)
where t.C = 'Jones'

This query asks us to print Jones' address. If we follow Algorithm 1 we find, naturally enough, that the expression E_5 involves "joining" only one relation LIVES, selecting for C = 'Jones', and projecting the result onto ADR. Notice how the question whether the tuple or tuples with C = 'Jones' in LIVES are dangling or not never comes up. Even if Jones does not appear in the hold or has relations, or for some other reason, the join of all the relations includes no tuple with C = 'Jones', our response to Q_1 has been the intuitively correct one. Unfortunately, when the hypergraph defining the structure of the universal relation is cyclic, as Fig. 1 is, Algorithm 1 can give intuitively wrong answers to queries, primarily, it appears, because dangling tuples are not always treated properly, but also because the minimal connection among the attributes of the query will not necessarily be embodied in the join of the expression E_5 of Algorithm 1.

Example 3: Consider, for the same database:

retrieve(t.BNK) (Q_2)
where t.C = 'Jones'

If we apply Algorithm 1, we find that the answer to query Q_3 is the set of banks where Jones has *both* a loan and an account. If we take for granted that the meaning of Q_3 is the set of banks at which Jones has *either* a loan or an account, and are not willing to incorporate dummy information about a loan when Jones opens an account at National, to make Q_3 come out correctly, then we conclude that Algorithm 1 does not handle Q_3 properly. The problem seems to be that if Jones has only a loan at National, the tuples that connect Jones and National are dangling.

Example 4: The following query is similar to Q_3:

retrieve(t.ACC) (Q_3)
where t.L = 4-326.

Query Q_3, like query Q_2, jumps across the diamond of Fig. 1. Despite this similarity, the intended meaning of Q_3 is not likely to be "Print the accounts that are either at the same bank as loan 4-326 or are owned by the customer who also holds loan 4-326." In fact, it isn't clear that Q_3 has any natural meaning. This example points up the fact that multiple paths connecting attributes can be a source of ambiguity for systems trying to deal with universal relations.

III. Maximal Objects

Evidently, we need some black magic. This magic must cause Q_2 and Q_3 to produce the correct answers, and it must be sufficiently powerful to distinguish between them, since they appear to be syntactically the same query. The magic might come from a wave of the semantic wand, such as the semantics of Codd [Co], where we worry about how attributes represent "entities" and "relationships," which are concepts rooted in

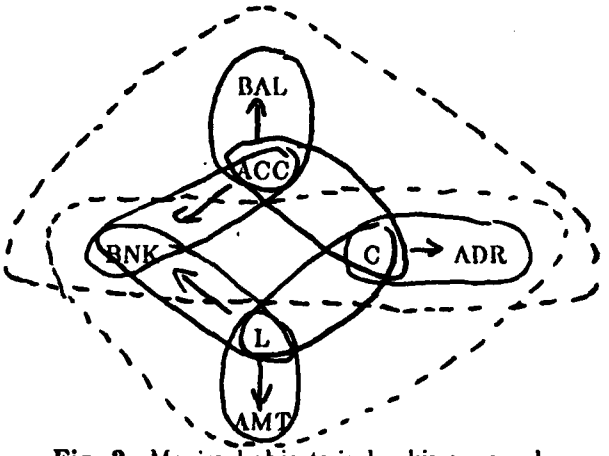


Fig. 2. Maximal objects in banking example.

what real world the database represents. We would prefer not to rely on semantic notions; rather, we would like purely syntactic ideas, because while computers tend to be incapable of dealing with semantics directly, they grind away at syntactic calculations quite amiably and with a deal of efficiency as well.

The extra syntactic notion we propose supplying, in addition to the FD's and objects, is a collection of sets of objects. Each set of objects is called a *maximal object*. Intuitively, the maximal objects are the largest sets of objects in which we are willing to navigate. For practical reasons, every object must be in at least one maximal object. We associate zero or more maximal objects with each tuple variable of a query—those maximal objects that contain all the attributes mentioned by the query in connection with that tuple variable.

Example 5: Figure 2 shows the two maximal objects we select for Fig. 1. The arrows represent FD's, but we ignore them for the moment; we shall use them when we explain how maximal objects might be formed. There are two maximal objects,

{ C-ADR, C-ACC, ACC-BAL, ACC-BNK } and
{ C-ADR, C-L, L-AMT, L-BNK }

which we call the *upper* and *lower* maximal objects, respectively. []

Query Q_2 has one tuple variable t, and its associated attributes are C and BNK. Both attributes are each contained in both maximal objects. We are willing to "navigate" within either maximal object. We take the meaning of Q_2 to be the union of the answers we get by evaluating the query over each maximal object.

When we evaluate with respect to the upper maximal object, we get the banks at which the customer has an account. That is, the optimization step in Algorithm 1 leaves us with the join of the ACCAT and OWN relations only. If Jones has an account at National, then we shall be told that fact when we apply Algorithm 1 to the upper maximal object, even if we have recorded in the database no address for Jones, no balance for any of his accounts, there is no loan by National to Jones, or any other problem arises that would technically cause Jones and National not to be related in the universal relation. The reader must judge for himself whether this is a reaonable response by a universal relation system, but we believe that to be the case.

When we evaluate Q_2 in the lower maximal object, we get the banks at which the customer has loans. Thus the interpretation of Q_2 is the set of banks at which the customer has either an account or loan, as we intuitively feel it should be.

Now consider query Q_3, which relates accounts and loans. These two attributes occur together in no maximal object. Thus an empty set of accounts should be produced by the system, or better, an error message saying the query cannot be processed or is ambiguous. We can still ask for the accounts held by the holder of 4-326:

retrieve(t.ACC) (Q_4)
where t.C=s.C and s.L=4-326

In this query, the attributes connected with t lie in the upper maximal object and those connected with s lie in the lower maximal object. Q_4 gives the intuitively correct result. A similar query would give us all the

accounts at the bank making loan 4-326. One anomaly we face concerns trivial queries like:

retrieve(t.C) (Q_5)
where t.C=t.C

That is, print all the customers. If we work in the upper maximal object, for example, Algorithm 1 tells us to answer the query by taking some relation whose scheme includes C and project it onto C. The result of Step (5) in Algorithm 1 is ambiguous; since the optimizer assumes weak equivalence of expressions is sufficient, i.e., the equivalence of expressions depends only on their values when the relations to which they apply are the projection of a universal relation. If that were the case, we would indeed get the same result whether we projected OWN or LIVES onto C.

In practice, dangling tuples may make the results of projections from OWN and LIVES onto C different. The maximal object concept we propose does not deal with this problem, and the solution probably lies in a modification of Algorithm 1 to take the union of all relations capable of producing a projection that the expression minimizes calls for. In that case, the response to Q_5 would be all customers mentioned in any of OWN, HOLD, and LIVES. Let us now formalize our notion of maximal objects. At the outset, the reader should be aware that maximal objects are not part of a "data model." Rather, they are parameters that influence a particular algorithm for query interpretation. Whether or not they produce the intuitively "correct" interpretation of queries in all situations is for the reader to judge. We can only give examples, such as Q_2 and Q_3, where the method seems to handle hard cases properly, and we shall give some intuition that supports our method, to be described later, for selecting maximal objects.

Definition: Let $m = \{ R_1, R_2, \ldots, R_k \}$ be a maximal object, Let $U_m = R_1 \cup R_2 \cup \cdots \cup R_k$, let $r_1, r_2, \ldots, r_k$ be the database relations for the objects $R_1, R_2, \ldots, R_k$, and let $u_m = r_1 \bowtie r_2 \bowtie \cdots \bowtie r_k$. If X is a set of attributes, the *connection in* m among the attributes of X, denoted $[X, m]$, is $\pi_X(u_m)$ if $X \subseteq U_m$ and $\emptyset$ otherwise.

If $M = \{ m_1, m_2, \ldots, m_q \}$ is the set of maximal objects for the database, the *connection* among the attributes in X *in the database* is given by

$$[X] = [X, m_1] \cup [X, m_2] \cup \ldots \cup [X, m_q]$$

This definition says to interpret queries as if we had a universal relation u given by:

$$u = u_{m_1} \cup u_{m_2} \cup \ldots \cup u_{m_q},$$

where each u_{m_i} has its tuples padded with nulls to be over the universal scheme.

This change in the interpretation of the database indicates we should modify Algorithm 1, in order to limit the range of navigation for tuple variables to maximal objects. We no longer construct a copy of the universal relation for each tuple variable. Instead, we find, for each tuple variable t, the (possibly empty) collection M_t of maximal objects that each include all the attributes associated with t in the query. For each m in M_t, we construct the relation u_m as in the definition above. We then let t range over all tuples in the union of all the u_m's such that m is in M_t. We formalize this construction in the next algorithm.

Algorithm 2: Given query Q mentioning tuple variables $t_1, t_2, \ldots, t_k$, and given maximal objects (sets of objects) $m_1, m_2, \ldots, m_q$, we convert Q to an algebraic expression as follows.

1. For each t_i, let $X_i = \{ B \mid t_i.B$ appears in $Q \}$. Let M_i be set of maximal objects M_j such that $X_i \subseteq U_j$, where U_j is the union of objects in M_j.
2. For each maximal object m_i, let J_i be the algebraic expression for the natural join of all the relations on objects in m_i.
3. For each tuple variable t_i, construct the algebraic expression K_i to be the union of expressions J_j over all j such that m_j is in M_i.
4. Let $E_2 = K_1 \times K_2 \times \cdots \times K_k$.
5. Construct E_3 by applying selection to E_2; construct E_4 from E_3 by applying projection, and construct E_5 from E_4 by optimization, exactly as in Steps 3, 4, and 5 of Algorithm 1. □

IV. Automatic Construction of Maximal Objects

We shall demonstrate a method by which the maximal objects of Fig. 2 might be obtained. In general, there is probably no substitute for the designer looking at the database and selecting the maximal objects

on the basis of what makes sense to him. Nevertheless, we can provide an algorithm, actually a variety of algorithms, for constructing maximal objects. The algorithms are based on the principle of Aho, Beeri and Ullman [ABU] that a join of relations "makes sense" if and only the join is lossless. While the sets of maximal objects so obtained may not give the intuitively correct answers to queries in all cases, they work in many cases, and are a starting point for the database designer.

The principle behind all the algorithms is that we start with a single object and "grow" it into a maximal object. We add objects to the maximal object being constructed so long as the join of the relations on the objects included s lossless. That is, we add a new object to the set under construction only if its relation joins losslessly with those for the objects already in the set. The difference among the algorithms for maximal objects lies in the strength of the rule used to deduce a lossless join. We assume procedure LOSSLESS(R, S) that looks at global information, such as FD's and the set of all objects, and returns true if and only if the join of the relations on objects R and S is lossless. We construct maximal objects as follows in Algorithm 3.

It should be noted that the step of that algorithm which finds a new object to add to the maximal object MO being formed is in a sense nondeterministic, in that we allow any eligible object to be chosen. In many cases, the predicate lossless will be *monotone*, in the sense that when $S \subseteq T$, we have LOSSLESS(R, S) implies LOSSLESS(R, T). Then, a candidate for inclusion in MO remains a candidate even if another candidate is chosen (selection is a "Church-Rosser" system), and the result of the algorithm is unique.

Algorithm 3:

```
MAXOBJ := ∅;
for each object do
    begin
        MO := {R};
        S := R;
        repeat
            find an object T not in MO such that LOSSLESS(T, S);
            MO := MO ∪ {T};
            S := S ∪ T
        until no such T is found;
        MAXOBJ := MAXOBJ ∪ {MO};
    end;
remove any set of objects from MAXOBJ that is a proper subset of another set;
```

□

We consider three versions of LOSSLESS:

1. LOSSLESS(R, S) = *true* if and only if $(R \cap S) \rightarrow R$ or $(R \cap S) \rightarrow S$ (the "FD's only" rule).
2. LOSSLESS(R, S) = *true* if and only if $(R \cap S) \twoheadrightarrow (R - S) \mid (S - R)$ (the "MVD" rule).
3. LOSSLESS(R, S) = *true* if and only if either
   a) $(R \cap S) \rightarrow R$ or $(R \cap S) \rightarrow S$, or
   b) $(R \cap S) \twoheadrightarrow (R - S) \mid (S - R)$ and *not* both of $(R - S) \rightarrow (R \cap S)$ and $(S - R) \rightarrow (R \cap S)$ are true.
   This rule is essentially the MVD rule, but prohibits navigation through "connection traps" [Co], such as from loan to bank to account in Fig. 2.

Rule 3 is more stringent than Rule 2, although more liberal than Rule 1. There is a seeming paradox with Rule 3. Not knowing about a vaid FD can allow it to create larger maximal objects than if we recognized the dependency. The motivation for Rule 3 is that in the absence of explicit directions to the contrary, we conjecture that a user does not want to navigate through a connection trap. That is, in Fig. 2, if the user had in mind a connection between loans and accounts, it would more likely be through customer than through bank. We shall let the reader make up his own mind whether that conjecture is true. Even if Rule 3 is the method adopted for constructing maximal objects, the user can always force a link to go through BNK by a two-tuple-variable query similar to Q_4.

Example 4: The maximal objects of Fig. 2 are constructed using the FD's-only rule. We obtain the upper maximal object by starting with object ACC-C. We can add ACC-BAL because of the FD ACC→BAL. We add C-ADR because of the FD C→ADR. BNK-ACC is added because of the FD ACC→BNK. The final maximal object includes attributes {ACC, C, BAL, ADR, BNK}.

Continuing the analysis of Fig. 2, the lower maximal object is created from C-L in a fashion quite analogous to the way the upper one was created. Starting with any object other than C-ACC or C-L yields a subset of the two maximal objects already constructed, so they are the only maximal objects produced.

Either of the other two rules for LOSSLESS yields the same maximal objects as in Fig. 2. []

The FD rule is easily seen to be monotone, while the other two rules are not, in general. However, the MVD rule is monotone in the important special case where all the MVD's follow from the given JD and FD's. Since all the given dependencies are full, it follows from the "chase" inference method of [MMS] that whenever we infer an embedded MVD $X \twoheadrightarrow Y \mid Z$, there is some full MVD $X \twoheadrightarrow Y' \mid Z'$, where $Y \subseteq Y'$ and $Z \subseteq Z'$, from which the embedded MVD follows.

In that case, rule (2) says LOSSLESS(R, S) is true if and only if $R \cap S$ multidetermines a set of attributes that includes $R - S$ but no attribute of $S - R$. In fact, since we start with a single JD, the test can be made in polynomial time [MSY].

There is another important fact about the MVD rule, which is brought out in the following theorem. This result says that when the JD defining the universal relation structure is acyclic, there is only one maximal object, and therefore Algorithms 1 and 2 treat queries the same way.

Theorem 1: If the given JD has an acyclic hypergraph [FMU] then every connected component of the hypergraph is a maximal object by the MVD rule.

Proof: We prove the result by induction on the number of edges in the hypergraph. The basis, one edge, is trivial. Each nontrivial connected component contains an edge (object) E with an attribute not present in any other edge and whose intersection with the union of the other edges is contained in one of those edges [BFMY]. If we remove E, the component remains acyclic, as can be proved easily using the Graham reduction test of [BFMY]. By induction on the number of edges, we claim there is an edge R in the component of E, with E removed, such that Algorithm 3, started with R, produces a maximal object with at least all the edges other than E in the component. Let the set of attributes in this maximal object be S. The given JD implies $(S \cap E) \twoheadrightarrow (E - S) \mid (S - E)$ [FMU]. Thus, E will be adjoined to S in Algorithm 3 to form a larger maximal object. []

V. Further Considerations and Conclusions

We have given three methods for constructing sets of maximal objects. Only experience in a variety of applications will show which method constructs maximal objects that give the best answers. Of course, a database designer is always free to include other maximal objects to make queries produce intuitively correct answers. For example, if the designer determined that the connection between loans and accounts for query Q_3 is always through customer, the maximal object { C-ACC, C-L } could be added. This maximal object would let Q_3 connect L and ACC through C.

Another way in which user-defined maximal objects can help is if there are embedded MVD's that do not follow from the given JD and FD's. For example, suppose that loans could be made by consortiums of banks, so the FD L→BNK no longer held. Then any of the three methods proposed for constructing maximal objects could find three: { C-ADR, C-ACC, ACC-BAL, ACC-BNK }, { L-BNK, L-AMT }, and { L-AMT, C-L, C-ADR }. That is, the lower maximal object gets split in two.

Now, the response of Algorithm 2 to Q_2 is to print only the banks at which Jones has an account, since only the upper maximal object includes all the attributes of t. We might feel that that answer is wrong, because Jones is still linked to all the banks to which he is related by being co-holder of a loan of which the bank is co-grantor. If one believes that to be the case, then one is really asserting the embedded MVD L↠BNK | C, that is, all banks granting a loan relate to all customers holding the loan.

Instead of declaring this embedded MVD, which leads to difficulties when we try to interpret queries by inferring lossless joins (see [MMS], e.g.) we would simply declare the lower maximal object, even though it doesn't follow from any of the construction rules we have proposed. This approach effectively substitutes maximal object declarations for certain collections of embedded MVD's, although it is not clear to what extent it enables us to ignore embedded MVD's entirely (except those that follow from the given JD or FD's), but the method appears promising.

The purpose of the universal relation user view for query evaluation, and the use of maximal objects therewith, is to remove the requirement of explicit knowledge of the database structure from the user. However, the sophisticated user could use knowledge of maximal objects to his advantage. One possibility

is to allow operands in the <condition>—clause of a query of the trivial form $t.A$, in order to require that tuple-variable t navigate only in maximal objects containing A. For example, a variant of query Q_2 is

retrieve(t.BNK) (Q_6)
where t.C='Jones' and t.ACC

Query Q_6 would be evaluated by letting t range only over the upper maximal object in Fig. 2. The query would produce just those banks where Jones has an account.

An alternative way to pass some navigation control to the user is by using aliases for some of the attributes to indicate in which maximal object the attribute is considered to lie. For example, we could have DEP (depositor) as an alias for C indicating the upper maximal object, and BOR (borrower) as an alias for C indicating the lower maximal object. With these aliases, the query

retrieve(t.BNK) (Q_7)
where t.DEP='Jones'

has the same meaning as query Q_6 above.

To conclude, we note as a consequence of Theorem 1, if the given JD has a connected, acyclic hypergraph, then the maximal object concept has no effect when the MVD rule is used. This should be the case, as a connected, acyclic hypergraph implies unique connections in the hypergraph among attributes. Thus, the universal relation idea by itself appears adequate when no ambiguity regarding navigation paths is present.

It is only when cycles occur, as in Fig. 2, that the need for maximal objects surfaces. We cannot be certain that maximal objects are more likely to give intuitively correct answers than the pure universal relation interpretation of queries (Algorithm 1). However, in Section IV we discussed an algorithm that finds maximal objects that naturally reflect the longest paths over which we can navigate through a particular object while maintaining a lossless join of the relations over which we travel. This origin for maximal objects lends a certain plausibility to their use.

Bibliography

[A] Aho, A. V., private communication, June, 1981.

[ABU] Aho, A. V., C. Beeri, and J. D. Ullman, "The theory of joins in relational databaes," *ACM Transactions on Database Systems* 4:3, pp. 297–314, Sept., 1979.

[ASU] Aho, A. V., Y. Sagiv, and J. D. Ullman, "Efficient optimization of a class of relational expressions," *ACM Transactions on Database Systems* 4:4, pp. 435–454, Dec., 1979.

[AP] Atenzi, P. and D. S. Parker, "Properties of acyclic database schemes: an analysis," *Proc. XP/2 Conf.*, Penn State, June, 1981.

[BFMY] "On the desirable properties of acyclic database schemes," manuscript in preparation.

[BG] Bernstein, P. A. and N. Goodman, "What does Boyce-Codd normal form do?," *Proc. International Conference on Very Large Data Bases*, pp. 245–259, 1980.

[Co] Codd, E. F., "Extending the database relational model to capture more meaning," *ACM Transactions on Database Systems* 4:4, pp. 397–434, Dec., 1979.

[FMU] Fagin, R., A. O. Mendelzon, and J. D. Ullman, "A simplified universal relation assumption and its properties," RJ2900, IBM, San Jose, Calif., 1980.

[HLY] Honeyman, P., R. E. Ladner, and M. Yannakakis, "Testing the universal instance assumption," *Information Processing Letters* 10:1, pp. 14–19, 1980.

[Ke] Kent, W., "Consquences of assuming a universal relation, IBM Tech. Rept., Dec., 1979.

[Kl] Klug, A., "Inequality tableaux," to appear in JACM.

[Ma1] Maier, D., *The Theory of Relational Databases*, Computer Science Press, 1982.

[Ma2] Maier, D., "Discarding the universal instance assumption," *Proc. XP/1 Conf.*, Stony Brook, N. Y., June, 1980.

[MMS] Maier, D., A. O. Mendelzon, and Y. Sagiv, "Testing implications of data dependencies," *ACM Transactions on Database Systems* 4:4, pp. 455-469, Dec., 1979.

[MSY] Maier, D., Y. Sagiv, and M. Yannakakis, "On the complexity of testing implications of functional and join dependencies," to appear in *JACM*.

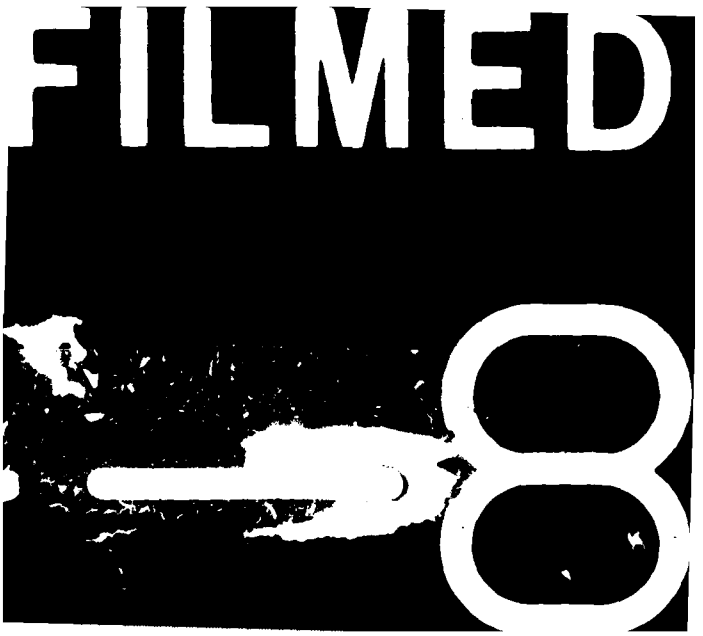
[MU] Maier, D. and J. D. Ullman, "Connections in acyclic hypergraphs," STAN-CS-758, Dept. of C. S., Stanford Univ., Stanford, Calif., 1981.

[S] Sagiv, Y., "Can we use the universal instance assumption without using nulls?," *ACM SIGMOD International Symposium on Management of Data*, pp. 108-120, 1981..

[Sc] Sciore, E., "Null values, updates, and normalization in relational databases," Ph. D. thesis, Princeton Univ., Princeton, N. J., 1980.

[SWKH] Stonebraker, M., E. Wong, P. Kreps, and G. Held, "The design and implementation of INGRES," *ACM Transactions on Database Systems* 1:3, pp. 189-222, Sept., 1976.

[Ul] Ullman, J. D., *Principles of Database Systems*, Computer Science Press, 1980.


FILMED